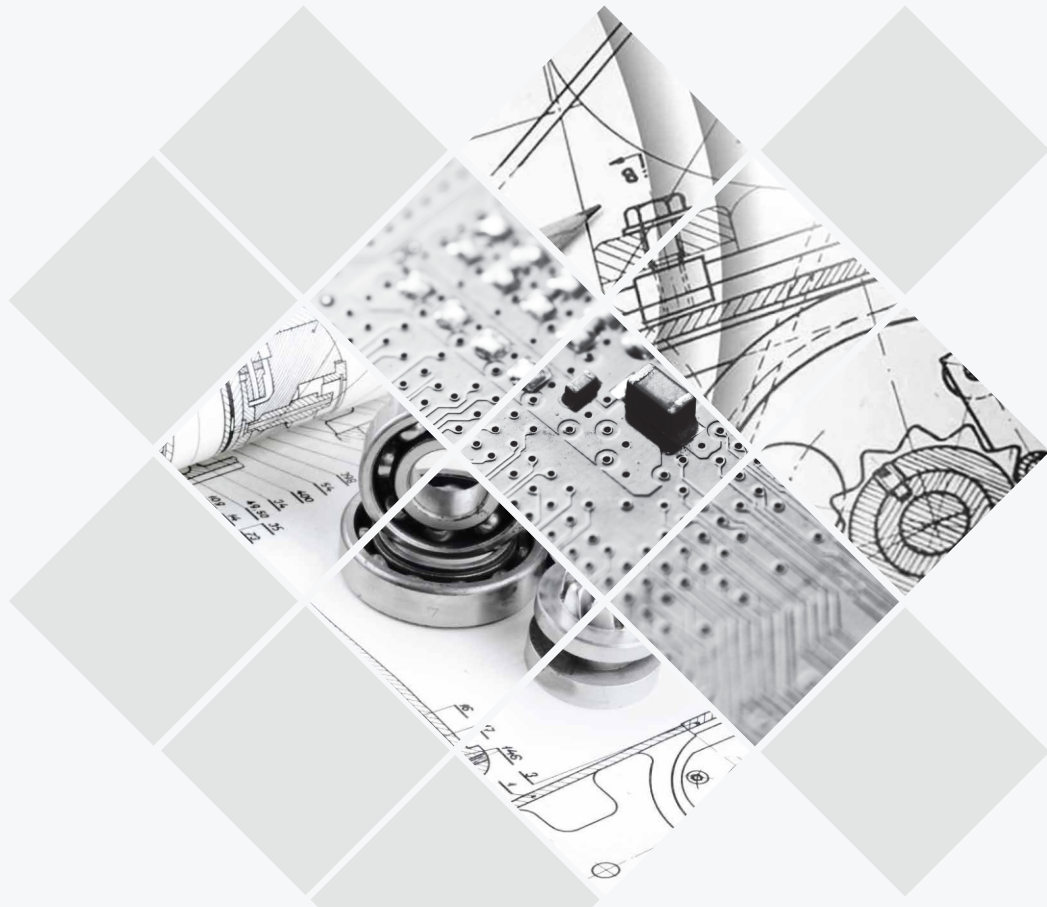# real-time face tracking with GPU acceleration

Real-time tracking of multiple faces in high resolution videos involve three basic tasks namely initialization, tracking and display.

Midhun M
Neethu K Chandran
Preetha Joy

QuEST Global

contents

Real-time Face Tracking with GPU Acceleration

Abstract

Fast and robust tracking of multiple faces is receiving increased attention from computer vision researchers as it finds potential applications in many fields like video surveillance and computer mediated video conferencing. Real-time tracking of multiple faces in high resolution videos involve three basic tasks namely initialization, tracking and display. Among these, tracking is quite compute intensive as it involves particle filtering that won't yield a real time performance if we use a conventional CPU based system alone. While looking forward a design that optimizes the system for an appreciable real-time performance, calls for the use of compute efficient platforms like GPU for compute intensive tasks along with conventional CPU. This paper discusses our heterogeneous design model which combines conventional programming for CPU efficient tasks and the nVIDIA CUDA for GP-GPU that implements the compute intensive tasks.

Keywords: Stream processing, GPU, GP-GPU, CUDA, Particle filtering, video tracking, real-time systems.

1. Introduction

Analysis of human faces has been an active research topic in computer vision and image processing for quiet a long time. This strong interest is driven by some promising applications such as surveillance and security monitoring, advanced human-machine interface, video conferencing and virtual reality. Generally speaking, major research areas include face detection, tracking and recognition, face animation, expression analysis, lip reading, etc. As the basis for all other related image analysis of human faces, face detection and tracking are of great importance.

Recently, there have been considerable research achievements in detection, recognition and tracking of human faces. A number of face detection and recognition algorithms can be found in a literature survey. Instead of detecting human faces in each frame independently, face tracking utilizes temporal correlation to locate them. The idea of introducing face tracking instead of doing repeated detection is to reduce the tracking time and produce real time output.

Our objective is to develop a real-time face tracker that tracks multiple faces simultaneously on subsequent video frames with maximum stability. For this a histogram based face tracking algorithm was chosen. Finally to achieve real time performance the algorithm was redesigned to a heterogeneous model using NVIDIA CUDA, that can run on a Graphics Processing Unit.

2. Methodology

As mentioned, a face tracker consists of two major parts detection and tracking. Detection will detect all the faces within a particular frame. The tracking part tracks the detected faces in the coming subsequent frames until the face leaves out of the frame. Detection is used for detecting new faces when it pops up onto the video frame. Hence forth detection need to be done only after n number of frames, based on an assumption that a new face will not pop up and leave all of a sudden. When faces in a frame are detected, faces which are already being tracked are allowed to continue the tracking process and newly detected faces are added on to the set of currently tracked faces and are tracked in the coming frames.

For face detection and face tracking there are a number of well defined proven algorithms available. The frontal face detection using Viola and Jones [3] boosting algorithm is used for detection. Face tracking which is our prime area of interest is implemented using Gray scale histogram method. This algorithm have many advantages over the other conventional algorithms, as it is robust to partial occlusion, rotation and scale invariant and calculated efficiently.

The entire operations of the application can be

Real-time Face Tracking with GPU Acceleration

grouped into initialization, tracking and display. Detecting new faces and the extraction of reference histogram of the grey scale image falls into the initialization phase. In tracking the detected faces are tracked. Tracking makes use of particle filtering based on condensation algorithm [6], which involves the particle selection, error diffusion, and weighting. Weighting in particle filtering is highly compute intensive portion and hence is our area of focus for performance enhancement. The display section handles the display of the output on to the screen. Before displaying, average of the particles from particle filtering is computed in-order to get the most appropriate position on the screen.
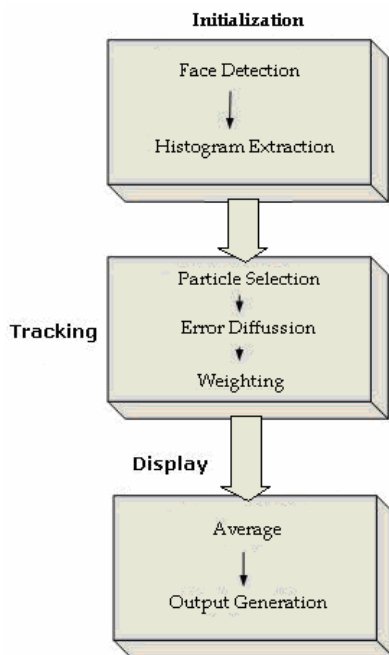

Figure 1. Block diagram of face tracking application

2.1 Particle Filtering

Particle filtering, also known as sequential Monte Carlo method (SMC) is sophisticated model estimation techniques based on simulation. Sampling importance resampling (SIR) is a very commonly used particle filtering algorithm. The Sampling-importance resampling (SIR) algorithm aims at drawing a random sample from a target distribution π. First, a sample is drawn from a proposal distribution q, and then from this a smaller sample is drawn with sample probabilities proportional to the importance ratios π/q. For a finite set of particles, the algorithm performance is dependent on the choice of the proposal distribution. The optimal proposal distribution is given as the target distribution. However, the transition prior is often used as importance function, since it is easier to draw particles (or samples) and perform subsequent importance weight calculations. Sampling Importance Resampling (SIR) filters with transition prior as importance function are commonly known as bootstrap filter and condensation algorithm [6]. The probabilistic approach of condensation algorithm provides significant robustness, as several possible states of the system are tracked at any given moment.

A common problem with this kind of probabilistic approach is its significant computational requirements. As the number of particles becomes large, the algorithmic calculation also increases. Fortunately, particle filters are easy to parallelize; they require high arithmetic throughput (as opposed to low latency), and have low global communication and storage costs. A good parallelization strategy can bring robustness of this algorithm to real-time applications.

2.2 Histogram-based Particle Filtering

A histogram is a method of quantizing the colors in the image. A color in the (R, G, B)-space consists of three values R, G and B, and in the histogram, the correlation between these values have to be retained. Therefore, it is not good enough to generate three histograms of the separate color channels (one for each color channel), but every (R, G, B)-triple will have to be "binned" as a whole. So the three channels can be combined together to get a single channeled gray scale image.

The proposed tracker employs the

Real-time Face Tracking with GPU Acceleration

Bhattacharya distance to update the priori distribution calculated by the particle filter. Bhattacharya coefficient is a popular method that uses histograms to correlate images. It defines a normalized distance among target histograms and histograms of candidate. It is expected that the maximum of this function gives the correct match.

3 Need For Performance Enhancement

For steady tracking of multiple faces in a real time video, we should be able to do tracking of faces on each and every frame without missing any frames in between. For this to happen in real rime, it is necessary that tracking of faces in one frame should be completed before capturing the next frame. Gray scale histogram based face tracking algorithm is highly efficient and robust, but at the same time it is highly compute intensive. So as the number of faces to be tracked in a frame increases the process of tracking faces in a single frame will consume more time and hence result in a low FPS video output. To achieve real time tracking of multiple faces using gray scale histogram method, it is indeed necessary to reduce the execution time of the tracking algorithm. A possible way of effectively reducing the time is to modularize the compute intensive portions in the algorithm and execute each module in parallel. Fortunately in histogram based tracking the compute intensive portions are very much parallelizable i.e. they have very low data interdependencies.

Once we are all set to parallelize the algorithm, we need an efficient stream processor which can execute the parallelized portions of the algorithm.Our interest mainly is bound across graphics chips, because they are currently the most powerful and cheap computational hardware available. These chips have gone from fixed-application peripherals to modern, powerful, and programmable general purpose processors. Unfortunately, the GPU uses an entirely different and unfamiliar programming model. So a very thorough know how of the programming model as well as the details of the underlying architecture is of at most importance for programming efficiently in GPU. NVIDIA Corp. provides now a full architecture based on the Stream processing model called CUDA (Compute Unified Device Architecture). CUDA has a C like programming model which helps a normal C/C++ programmer to write programs that can harness the immense power of the GPU.

4 NVIDIA CUDA programming model

NVIDIA CUDA (Compute Unified Device Architecture) is a hardware and software architecture that allows the GPU to be viewed as a data-parallel computing device that operates as a coprocessor to the main CPU (the host).

At the hardware level, the GPU is a collection of multiprocessors, with several processing elements in each. Each processor in the multiprocessor executes the same instruction in every cycle. Each can operate on its own data, which makes each a SIMD processor. Communication between multiprocessors is only through the device memory, which is available to all cores of the multiprocessors. The processing elements of a multiprocessor can synchronize with one another, but there is no direct synchronization mechanism between the multiprocessors. The GPUs provides only single-precision floating point numbers and 32-bit integers on native numeric data types, though this may change in near future.

For the programmer, the CUDA consists of a collection of threads running in parallel and all threads execute a single program called the kernel. Kernels have a Single Program Multiple Data (SPMD) programming model that allows limited divergence in execution. A part of the application that is executed many times, but independently on different elements of a dataset, can be isolated into a kernel that is executed on the GPU in the form of many different threads.

Real-time Face Tracking with GPU Acceleration

Kernels run on a grid, which is an array of blocks; and each block is an array of threads.

The kernel is the core code to be executed on each thread. Using the thread and block IDs, each thread can perform the kernel task on different data. Since the device memory is available to all the threads, it can access any memory location. The performance improves with the use of shared memory which can be accessed in a single clock cycle. In contrast, the global or device memory access takes 200-400 cycles. Read only texture memory optimized for 2-D texture fetch and constant memory assigned by the CPU are also available. Their access is slow but the internal caching mechanism reduces the effective access times for coherent access. The hardware architecture allows multiple instruction sets to be executed on different multiprocessors. The current CUDA programming model, however, cannot assign different kernels to different multiprocessors, though this may be simulated using conditionals.
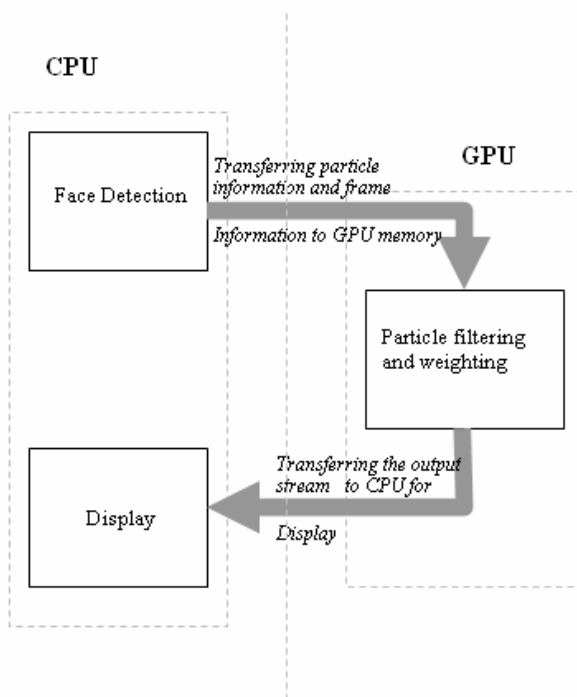

Figure 2. Architectural overview of the tracker using GPU.

Algorithm:

1. Capture the input frame.
2. Detect faces in the frame.
3. Check if the detected face is a new face. If the face is not newly detected go to step 6.
4. Do template extraction. Find histogram of the face.
5. Do particle initialization with current location and weight as Zero
6. Copy the particle information corresponding to all faces and frame information, to the GPU memory for executing the steps below.
7. Do particle selection, add Gaussian noise.
8. Calculate Histogram for each particle.
9. Calculate the Bhattacharya similarity coefficient between the reference histogram and calculated histogram.
10. Assign weight for each particle according to Bhattacharya similarity coefficient.
11. Copy back the weight information of each particle to the CPU.
12. Sort the particle according to weight.
13. Draw rectangle around the face along the particle with maximum weight and display on screen.

5 CUDA Implementation Details

Our application uses CUDA implementation in the weight calculation step of the particle filtering algorithm. Weight calculation is the most time consuming portion in the algorithm. And it can be

Real-time Face Tracking with GPU Acceleration

made to execute in parallel on the GPU. The following are the steps involved in the GPU implementation of weight calculation.

1. At a particular frame (frame 0), the reference histogram is copied to the device memory. This input stream doesn't change during the tracking process, so it is kept in device memory and used in each iteration of the algorithm.
2. For each input frame (frame t), the frame buffer and the particle information is copied to the device memory.
3. The kernel function is invoked and is executed over the blocks and threads. Each thread calculates the gray-scale histogram based on corresponding particle information. The weight of the particle is calculated from the calculated histogram and the reference histogram using the Bhattacharya distance. The result is then stored in the device memory.
4. The auto regressive dynamics used for the new state calculation is also done inside the kernel. The result of the calculation is used to update the device buffer with the new particle information.
5. The host recovers from device memory the output stream containing the weight of every particle and the updated particle state. The algorithm continues its normal flow of execution until a new video frame and stream of particles requires processing.

As a first level of implementation we did only a particle level parallelism. Accordingly only particles corresponding to one face was processed simultaneously in parallel. All the faces were iterated from the CPU one after the other, to compute the weight corresponding to the particles of each face. To achieve better performance, constant memory and texture memory were used instead of the device global memory. Texture memory and constant memory are cached memory and can hence account for performance enhancement if used judiciously. The Gaussian noise used for state transition

remains constant for the entire application, so it is copied only once and is stored in device's constant memory. But the input stream which varies with each frame in the tracking process is copied to the device buffer and is stored in texture memory for each iteration of the tracking process.

As an enhancement from the initial implementation, a face level parallelism was then implemented to utilize the maximum power of the GPU. If we are doing only particle level parallelism, we will only be using very small number of the underlying processing cores in the GPU. In order to overcome this shortcoming we parallelize in a face cum particle level. In this implementation all the particles corresponding to all the faces in a frame are executed simultaneously and will account for using a large number of processing cores of the GPU. Since there are no dependencies between different particles available it doesn't raise much trouble in the implementation. According to this we transfer the entire frame and all the particle information corresponding to all the tracking faces at once into the GPU and process all of them simultaneously in parallel. A face cum particle level parallelism resulted in an immense performance gain over the particle level parallism. According to the current implementation as the number of faces in a frame increases the performance of the GPU implementation goes on increasing when mapped to its CPU implementation.

6 Experimental results

Figure 3. A normal tracking in progress.

Real-time Face Tracking with GPU Acceleration

Figure 4. Tracking of rotated faces.

Figure 5. Tracking of multiple faces.

7 Performance figures

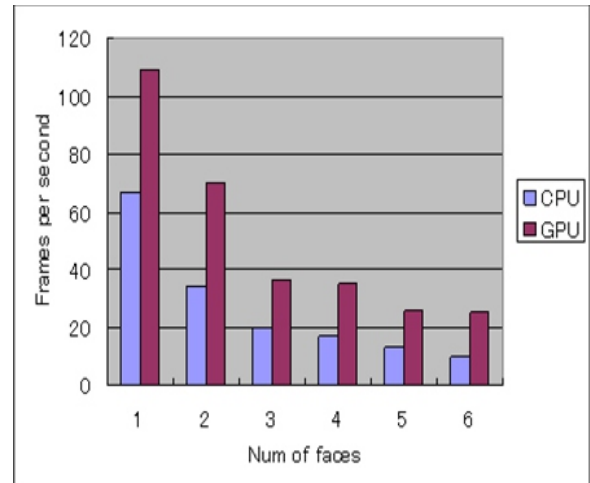|	GPU	CPU
Number of Frames tracked per second for 5 faces	36	20

Figure 6: The performance gain in terms of FPS for CPU Vs GPU as the number of faces increases

8 Future Enhancements

As of now the video tracking serves the purpose of tracking faces in video. To add as an enhancement to the existing tracker we plan to give support for face recognition. In face recognition it would be possible to identify a particular face from a group of people. This can be done by cross checking the detected faces with the face to be recognized which we have in our database, and then tracking the matching face. This can be of high importance in video surveillance and security as it helps to recognize a person among a group of people and track his movement. Regarding the performance improvement, we are planning to port the particle sorting step to GPU, which we hope will give considerable performance gain.

9 References

[1] NVIDIA, CUDA Programming Guide 1.1

[2] Lozano O. L., Otsuka K. (2008) Real-time Visual Tracker by Stream Processing, In Journal of Signal Processing Systems, July 2008 accessible at:

Real-time Face Tracking with GPU Acceleration

http://www.springerlink.com/content/pk22n1
632859082k/fulltext.pdf

[3] Viola, P., & Jones, M. (2001). Rapid object
detection using a boosted cascade of simple
features. In Proc. of the IEEE computer
society conference on computer vision and
pattern recognition (Vol. 1, pp. 511–518).

[4] Ahlberg, J. (2001). Candide-3 – an updated
parameterized face. Technical report, Dept.
of Electrical Engineering, Linköping
University.

[5] Arulampalam, S., Maskell, S., Gordon, N.
J., & Clapp, T. (2002). A tutorial on particle
filters for on-line nonlinear/non-Gaussian
Bayesian tracking. IEEE Transactions of
Signal Processing, 50(2), 174–188,
February.

[6] Condensation – conditional density
propagation for visual tracking, Isard and
Blake, Int. J.Computer Vision, 1998

BORN TO ENGINEER